IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

		ATTY. DOCKET NO.: **AUS920031085US1**
IN RE APPLICATION OF:	§	
	§	
SCOTT J. BROUSSARD	§	EXAMINER: **HWA, SHYUE JIUNN**
	§	
SERIAL NO.: **10/798,910**	§	CONFIRMATION NO.: **6996**
	§	
FILED: **MARCH 11, 2004**	§	ART UNIT: **2163**
	§	
FOR: **METHOD, SYSTEM AND ARTICLE FOR DETECTING CRITICAL MEMORY LEAKS CAUSING OUT-OF-MEMORY ERRORS IN JAVA SOFTWARE**	§ § § § §	

APPEAL BRIEF UNDER 37 C.F.R. §1.192

Mail Stop Appeal Briefs - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This Brief is submitted in support of the Appeal of the Examiner's final rejection of Claims 1-27 in the above-identified application. A Notice of Appeal was filed in this case on March 11, 2008 and received in the United States Patent and Trademark Office on March 11, 2008. Please charge the fee of $510.00 due under 37 C.F.R. §41.20(b)(2) for filing the brief, as well as any additional required fees, to **IBM Corporation Deposit Account No. 09-0447.**

REAL PARTY IN INTEREST

The real party in interest in the present Application is International Business Machines Corporation, the Assignee of the present application as evidenced by the Assignment set forth at reel 015681, frame 0478.

RELATED APPEALS AND INTERFERENCES

There are no Appeals or Interferences known to Appellants, the Appellants' legal representative, or assignee, which would be directly affected or have a bearing on the Board's decision in the present Appeal.

STATUS OF CLAIMS

Claims 1-27 are pending in the application and stand finally rejected by the Examiner as noted in the Final Action dated December 11, 2007. The rejections of Claims 1-27 are appealed.

STATUS OF AMENDMENTS

No amendments to the claims have been made subsequent to the final rejections that lead to this appeal. The claims presented herein are in the form in which they were presented to the Examiner prior to the final rejection.

SUMMARY OF THE CLAIMED SUBJECT MATTER

The invention recited in Claim 1 provides a computer-implemented method for assisting the detection of critical memory leaks in a software program. According to this method, an amount of available memory for the software program is monitored during execution of the software program (FIG 2., ref. nos. 204, 206; ¶7, ll 3-5; ¶55, ll 8-16). A determination is made whether the amount of available memory for the software program is less than a predetermined amount (FIG 10, blocks 1006 and 1008; ¶7, ll 6-11; ¶55, ll 11-14). In response to such determination, a current stack walkback of each object currently referenced by the software

program is stored prior to the available memory dropping below an amount necessary to store the current stack walkback, where the current stack walkback assists in the detection of a critical memory leak during execution of the software program (FIG 2., ref. nos. 202, 204, 206, 208, 212, 214, 226; FIG. 7, block 716; FIG. 10., ref. nos. 1008, 1012; ¶7, ll 11-16; ¶20, ll 8-17; ¶29, ll 22-28; ¶50, ll 5-18).

Claims 2 and 9 depend from claim 1. Claim 2 recites the steps of monitoring a specified one or more analysis properties of the objects referenced by the software program, where the one or more specified-analysis properties consists of at least one of an object's age and an object's instance count (¶29, ll 10-17; ¶37-38; ¶41, ll 1-9). A determination is made whether any analysis property of the objects being referenced following a garbage collection process exceeds a predetermined limit for such analysis property, where the predetermined limit for an object's age is an object age limit and the predetermined limit for an object's instance count is an object instance count growth value (¶¶37-38; ¶47, ll 6-13). Any objects determined to have one or more analysis properties that exceeds that property's predetermined limit is identified (¶47, ll 6-25). Claim 9 recites that the software objects are Java objects (FIG. 2, ref nos. 202, 204; ¶23, ll 8-10).

Claims 3-6 depend from claim 2. Claim 3 recites the step of calculating an object's age by timing a current period starting when the respective object was instantiated (¶47, ll 6-15). Claim 4 recites the step of calculating an object's instance count growth as the magnitude of growth of an object instance count over a given time period (¶47, ll 6-22). Claim 5 recites that the monitoring step includes monitoring objects within a class designated for monitoring (¶29, ll 17-19). Claim 6 recites the step of performing the current stack walkbacks for the identified objects (¶20, ll 8-12; ¶56, ll 1-5).

Claim 7 depends from claim 6. Claim 7 recites the step of generating a statistics report (FIG. 8, block 816) (FIG. 2, ref. no. 208) including current stack walkbacks for the identified objects (¶29, ll 17-24).

Claim 8 depends from claim 7. Claim 8 recites the step of generating a web interface for user viewing of the statistics report (FIG. 9, block 910) at a computer display (¶54, ll 7-10).

Claim 10 recites a system that is implemented using a computer (FIG. 1, ref. no. 100) for assisting the detection of critical memory leaks in a software program. The system includes a means for monitoring an amount of available memory for the software program during execution

of the software program (FIG 2., ref. nos. 204, 206; ¶7, ll 3-5; ¶55, ll 8-16). Moreover, the system includes a means for determining if the amount of available memory for the software program is less than a predetermined amount (FIG 10, blocks 1006 and 1008; ¶7, ll 6-11; ¶55, ll 11-14). In response to the determination by the determining means, a current stack walkback of each object currently referenced by the software program prior to the available memory dropping below an amount necessary to store the current stack walkback is stored, where the current stack walkback assists in the detection of critical memory leak during execution of the software program (FIG 2., ref. nos. 202, 204, 206, 208, 212, 214, 226; FIG. 7, block 716; FIG. 10., ref. nos. 1008, 1012; ¶7, ll 11-16; ¶20, ll 8-17; ¶29, ll 22-28; ¶50, ll 5-18).

Claims 11 and 18 depend from claim 10. Claim 11 recites a means for monitoring a specified one or more analysis properties of the objects referenced by the software program, where the one or more specified-analysis properties consists of at least one of an object's age and an object's instance count (¶29, ll 10-17; ¶37-38; ¶41, ll 1-9). The system includes a means for determining if any analysis property of the objects being referenced following a garbage collection process exceeds a predetermined limit for such analysis property, where the predetermined limit for an object's age is an object age limit and the predetermined limit for an object's instance count is an object instance count growth value (¶¶37-38; ¶47, ll 6-13). The system also includes a means for identifying software objects determined to have one or more analysis properties that exceeds that property's predetermined limit (¶47, ll 6-25). Claim 18 recites that the software objects are Java objects (FIG. 2, ref nos. 202, 204; ¶23, ll 8-10).

Claims 12-15 depend from claim 11. Claim 12 recites a means for calculating an object's age by timing a current period starting when the respective object was instantiated (¶47, ll 6-15). Claim 13 recites a means for calculating object instance count growth as the magnitude of growth of an object's instance count over a given time period (¶47, ll 6-22). Claim 14 recites that the means for monitoring includes a means for monitoring objects within a class designated for monitoring (¶29, ll 17-19). Claim 15 recites a means for performing the current stack walkbacks for the identified objects (¶20, ll 8-12; ¶56, ll 1-5).

Claim 16 depends from claim 15. Claim 16 recites a means for generating a statistics report (FIG. 8, block 816) (FIG. 2, ref. no. 208) including current stack walkbacks for the identified objects (¶29, ll 17-24).

Claim 17 depends from claim 16. Claim 17 recites a means for generating a web interface for user viewing of the statistics report (FIG. 9, block 910) at a computer display (¶54, ll 7-10).

Claim 19 recites an article of manufacture (¶58) comprising a machine-readable medium including program logic embedded therein for assisting the detection of critical memory leaks in a software program that causes control circuitry in a data processing system to perform the steps of monitoring an amount of available memory for the software program during execution of the software program (FIG 2., ref. nos. 204, 206; ¶7, ll 3-5; ¶55, ll 8-16). Other steps that are performed include determining if the amount of available memory for the software program is less than a predetermined amount (FIG 10, blocks 1006 and 1008; ¶7, ll 6-11; ¶55, ll 11-14). In response to the determining step, a current stack walkback of each object currently referenced by the software program prior to the available memory dropping below an amount necessary to store the current stack walkback is stored, where the current stack walkback assists in the detection of critical memory leak during execution of the software program (FIG 2., ref. nos. 202, 204, 206, 208, 212, 214, 226; FIG. 7, block 716; FIG. 10., ref. nos. 1008, 1012; ¶7, ll 11-16; ¶20, ll 8-17; ¶29, ll 22-28; ¶50, ll 5-18).

Claims 20 and 27 depend from claim 19. Claim 20 recites monitoring a specified one or more analysis properties of the objects referenced by the software program, where the one or more specified-analysis properties consists of at least one of an object's age and an object's instance count (¶29, ll 10-17; ¶37-38; ¶41, ll 1-9). A determination is made whether any analysis property of the objects being referenced following a garbage collection process exceeds a predetermined limit for such analysis property, where the predetermined limit for an object's age is an object age limit and the predetermined limit for an object's instance count is an object instance count growth value (¶¶37-38; ¶47, ll 6-13). Any objects determined to have one or more analysis properties that exceeds that property's predetermined limit is identified (¶47, ll 6-25). Claim 27 recites that the software objects are Java objects (FIG. 2, ref nos. 202, 204; ¶23, ll 8-10).

Claims 21-24 depend from claim 20. Claim 21 recites the step of calculating an object's age by timing a current period starting when the respective object was instantiated (¶47, ll 6-15). Claim 22 recites the step of calculating an object's instance count growth as the magnitude of growth of an object instance count over a given time period (¶47, ll 6-22). Claim 23 recites that

the monitoring step includes monitoring objects within a class designated for monitoring (¶29, ll 17-19). Claim 24 recites the step of performing the current stack walkbacks for the identified objects (¶20, ll 8-12; ¶56, ll 1-5).

Claim 25 depends from claim 24. Claim 25 recites the step of generating a statistics report (FIG. 8, block 816) (FIG. 2, ref. no. 208) including current stack walkbacks for the identified objects (¶29, ll 17-24).

Claim 26 depends from claim 25. Claim 26 recites the step of generating a web interface for user viewing of the statistics report (FIG. 9, block 910) at a computer display (¶54, ll 7-10).

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

I. Is the Examiner's rejection of Claims 1, 6, 7, 10, 15, 16, 19, 24 and 25 under 35 U.S.C. § 102(e), as being anticipated by *Cantrill* (U.S. Patent No. 6,523,141 B1) (hereinafter *Cantrill*) well founded?

II. Is the Examiner's rejection of Claims 2-5, 8, 11-14, 17, 20-23 and 26 under 35 U.S.C. § 103(a) as being unpatentable over *Cantrill*, in view of *Schumacher* (U.S. Patent Application No. 2005/0076184 A1) (hereinafter *Schumacher*) and further in view of *Fu* (U.S. Patent 7,089,460 B2) (hereinafter *Fu*) well founded?

III. Is the Examiner's rejection of Claims 9, 18 and 27 under 35 U.S.C. § 103(a) as being unpatentable over *Cantrill*, in view of *Cirne* (U.S. Patent Application No. 2004/0078540 A1) (hereinafter *Cirne*) well founded?

ARGUMENTS

I. The rejection of Claims 1, 6, 7, 10, 15, 16, 19, 24 and 25 under 35 U.S.C. § 102(e), as being anticipated by *Cantrill* is not well founded and should be reversed.

On page 3 of the Final Office Action, claims 1, 6, 7, 10, 15, 16, 19, 24 and 25 were rejected under 35 U.S.C. § 102(e) as being anticipated by *Cantrill.* The Examiner's rejection should be reversed because *Cantrill* does not teach, show, or suggest each claimed feature. In particular, nothing in *Cantrill* teaches, shows, or suggests:

(i) monitoring an amount of available memory for the software program during execution of the software program;

(ii) determining if the amount of available memory for the software program is less than a predetermined amount; and

(iii) in response to such determination, storing a current stack walkback of each object currently referenced by the software program prior to the available memory dropping below an amount necessary to store the current stack walkback, wherein the current stack walkback assists in the detection of a critical memory leak during execution of the software program.

(i) *Cantrill* fails to teach, show, or suggest "monitoring an amount of available memory for the software program during execution of the software program".

As a basis for the § 102(e) rejection, the Examiner cites col. 9, lines 42-48 of *Cantrill* as disclosing "monitoring an amount of available memory for the software program during execution of the software program". The cited passage of *Cantrill* discloses that in one embodiment, iterating or walking through a root set may involve first walking through all module data, then walking through all available thread stacks. Alternatively, *Cantrill* discloses that walking through the root set may involve walking through all available thread stacks prior to walking through all available buffers. However, the method, system, and computer program product disclosed in *Cantrill* involve the <u>post-mortem</u> detection of memory leaks. Thus, the step disclosed in col. 9, lines 42-48 of *Cantrill* of:

> "...iterating or walking through the root set may involve first walking through all module data, and then walking through all available thread stacks. Alternatively, walking

through the root set may involve walking through all available thread stacks prior to walking through all available thread stacks prior to walking through all available buffers",

has been taken out of context in view of the general idea taught by *Cantrill* that the detection of the memory leaks is performed after a computer system has crashed.

As such, the monitoring of available memory during an execution of a software program does not occur during execution of the software program in *Cantrill.*

Cantrill defines its post-mortem detection process:

> "A post-mortem memory leak detection process enables a system crash dump, or a file which essentially contains an image of the operating system of a system which has failed, to be used to locate memory leaks. By using data sections associated with the kernel as a root set, in addition to information contained in the crash dump file, a suitable garbage collection algorithm, e.g., a mark and sweep garbage collection process, may be used to identify memory leaks." (col. 4, lines 51-59)

Thus, *Cantrill* is not concerned with monitoring the amount of available memory, nor in determining if the amount of available memory for a software program is less than a predetermined amount, since the analysis steps performed in *Cantrill* would occur after a system crash has occurred.

(ii) *Cantrill* fails to teach, show, or suggest "determining if the amount of available memory for the software program is less than a predetermined amount".

Examiner contends that *Cantrill* teaches Appellant's step of "determining if the amount of available memory for the software program is less than a predetermined amount". In support of his argument, Examiner cites col. 9, lines 48-52 of *Cantrill*, which states:

> "It should be appreciated that in some cases, walking through module data sections may involve walking through available buffers and available thread stacks such that some buffers are walked through before some thread stacks, and vice versa."

However, Examiner's argument is a tenuous one, since there is nothing in the above quoted passage of *Cantrill* which could be construed as teaching or suggesting the step of determining if the amount of available memory for a software program is less than a predetermined amount, as disclosed in claim 1 of Appellant's claimed invention. At best, the passage in *Cantrill* merely

teaches that sections of data contained in available buffers and available thread stacks are walked through in a particular order.

(iii) *Cantrill* fails to teach, show, or suggest "in response to such determination, storing a current stack walkback of each object currently referenced by the software program prior to the available memory dropping below an amount necessary to store the current stack walkback, wherein the current stack walkback assists in the detection of a critical memory leak during execution of the software program".

Moreover, Examiner contends in the Final Office Action that *Cantrill* teaches the "...detection of critical memory leaks during execution of the software program." Examiner is basing his contention on *Cantrill*, col. 5, lines 3-5, which states in part that:

> "A post-mortem memory leak detection process allows run-time considerations that are typically associated with detecting memory leaks..."

However, Examiner has taken the above partial citation in *Cantrill* completely out of context. In this regard, Examiner fails to also include the second half of the quoted sentence, subsequent lines 6-8 of col. 5 of *Cantrill*, which in combination with cited lines 3-5 teaches quite the opposite:

> "A post-mortem memory leak detection process allows run-time considerations that are typically associated with detecting memory leaks, e.g., considerations of system overhead and time, <u>may be substantially avoided. In other words, a post-mortem memory leak detection process may be performed substantially off-line</u>." (underlined emphasis added for lines 6-8)

In view of the fact that the Examiner has misinterpreted *Cantrill*, Appellant's claims overcome *Cantrill* since *Cantrill* fails to anticipate, show, or suggest any of the steps recited in Appellant's independent claim 1, which specifically recite "during the execution of the software program."

In light of the preceding argument, Appellant submits that independent Claim 1, similar claims 10 and 19 and all dependent claims are not anticipated by *Cantrill*, and the Examiner's rejection under 35 U.S.C. § 102(e) should be reversed.

II. The rejection of dependent Claims 2-5, 8, 11-14, 17, 20-23 and 26 under 35 U.S.C. § 103(a), as being unpatentable over *Cantrill* in view of *Schumacher*, and further in view of *Fu* is not well founded and should be reversed.

On page 7 of the Final Office Action, claims 2-5, 8, 11-14, 17, 20-23 and 26 were rejected under 35 U.S.C. § 103(a) as being unpatentable over *Cantrill* in view of *Schumacher*, and further in view of *Fu*. The Examiner's rejection should be reversed because neither *Cantrill*, *Schumacher*, *Fu*, nor their combination teaches, shows, or suggests all of the claimed features recited in the abovementioned dependent claims.

First, none of the references, either individually or in combination, teach, show, or suggest the limitations recited in dependent claims 2-5, 8, 11-14, 17, 20-23 and 26, which include the limitations of base claims 1, 10, and 19. Examiner's arguments against the patentability of the aforementioned claims hinge upon the teachings of *Cantrill*. However, for the reasons argued in section **I.** of this appeal brief, *Cantrill* fails to teach, show, or suggest the steps of:

(i) monitoring an amount of available memory for the software program during execution of the software program;
(ii) determining if the amount of available memory for the software program is less than a predetermined amount; and
(iii) in response to such determination, storing a current stack walkback of each object currently referenced by the software program prior to the available memory dropping below an amount necessary to store the current stack walkback, wherein the current stack walkback assists in the detection of a critical memory leak during execution of the software program.

Moreover, neither *Schumacher* nor *Fu* make up for the deficiencies present in *Cantrill*. In particular, *Schumacher* and *Fu* also fail to teach, show, or suggest element (iii), which stores a current stack walkback of each object. In fact, *Schumacher* and *Fu* fail to even make any mention of stack walkbacks or stack traces.

In view of the foregoing, Appellant submits that dependent claims 2-5, 8, 11-14, 17, 20-23 and 26 are not obvious in view of the combination of *Cantrill*, *Schumacher*, and *Fu* and the Examiner's rejection under 35 U.S.C. § 103(a) should be reversed.

III. The rejection of dependent Claims 9, 18, and 27 under 35 U.S.C. § 103(a), as being unpatentable over *Cantrill* in view of *Cirne* is not well founded and should be reversed.

On page 13 of the Final Office Action, claims 9, 18, and 27 were rejected under 35 U.S.C. § 103(a) as being unpatentable over *Cantrill* in view of *Cirne*. The Examiner's rejection should be reversed because neither *Cantrill*, *Cirne*, nor their combination teaches, shows, or suggests all of the claimed features recited in the abovementioned dependent claims.

First, none of the references, either individually or in combination, teach, show, or suggest the limitations recited in dependent claims 9, 18, and 27, which include the limitations of base claims 1, 10, and 19. Examiner's arguments against the patentability of the aforementioned claims hinge upon the teachings of *Cantrill*. However, for the reasons argued in section **I.** of this appeal brief, *Cantrill* fails to teach, show, or suggest the steps of:

(i) monitoring an amount of available memory for the software program during execution of the software program;
(ii) determining if the amount of available memory for the software program is less than a predetermined amount; and
(iii) in response to such determination, storing a current stack walkback of each object currently referenced by the software program prior to the available memory dropping below an amount necessary to store the current stack walkback, wherein the current stack walkback assists in the detection of a critical memory leak during execution of the software program.

Moreover, *Cirne* fails to make up for the deficiencies present in *Cantrill*. In particular, *Cirne* also fails to teach, show, or suggest element (i), the monitoring of an amount of available memory for a software program during execution of a software program. The only monitoring that occurs in *Cirne* is the "tracking of growth patterns of groups of stored items" (*Cirne*, p. 2, ¶0015) and the "tracking of growth patterns in collection instances" (*Cirne*, p. 2, ¶0020). Such

types of monitoring taught in *Cirne* are unrelated to monitoring the amount of available memory for a software program.

In addition, *Cirne* teaches the creation of a stack trace for a collection object in response to a time out period, which is different to what is recited in element (iii) of Appellant's claim 1 and related claims 10 and 19. According to Appellant's claims 1, 10, and 19, a current stack walkback is stored in response to a determination that the amount of available memory for a software program is less than a predetermined amount.

In view of the foregoing, Appellant submits that dependent claims 9, 18, and 27 are not obvious in view of the combination of *Cantrill* and *Cirne* and the Examiner's rejection under 35 U.S.C. § 103(a) should be reversed.

CONCLUSION

Appellant believes that the foregoing arguments clearly demonstrate that:

I. *Cantrill* does not anticipate claims 1, 6, 7, 10, 15, 16, 19, 24 and 25 under 35 U.S.C. § 102(e);

II. The combination of *Cantrill*, in view of *Schumacher*, and further in view of *Fu* does not render claims 2-5, 8, 11-14, 17, 20-23 and 26 unpatentable under 35 U.S.C. § 103(a); and

III. The combination of *Cantrill*, in view of *Cirne* does not render claims 9, 18 and 27 unpatentable under 35 U.S.C. § 103(a)

Appellant has made diligent efforts to advance the prosecution of this application by pointing out with specificity the manifest error in the Examiner's rejections, and the claim language that renders the invention patentable over the prior art of record. Accordingly, Appellant respectfully requests all of the rejections of claims 1-27 be reversed, and a Notice of Allowance for claims 1-27 is courteously solicited.

Respectfully submitted,

Carlos E. Amorin
Reg. No. 48,379
DILLON & YUDELL LLP
8911 N. Capital of Texas Highway
Suite 2110
Austin, Texas 78759
512-343-6116

ATTORNEY FOR APPELLANT(S)

CLAIMS APPENDIX

1. A computer-implemented method for assisting the detection of critical memory leaks in a software program, the method comprising the steps of:

monitoring an amount of available memory for the software program during execution of the software program;

determining if the amount of available memory for the software program is less than a predetermined amount; and

in response to such determination, storing a current stack walkback of each object currently referenced by the software program prior to the available memory dropping below an amount necessary to store the current stack walkback, wherein the current stack walkback assists in the detection of a critical memory leak during execution of the software program.

2. The method according to claim 1, further comprising the steps of

monitoring a specified one or more analysis properties of the objects referenced by the software program, wherein the one or more specified-analysis properties consists of at least one of an object's age and an object's instance count;

determining if any analysis property of the objects being referenced following a garbage collection process exceeds a predetermined limit for such analysis property, wherein the predetermined limit for an object's age is an object age limit and the predetermined limit for an object's instance count is an object instance count growth value; and

identifying any objects determined to have one or more analysis properties that exceeds that property's predetermined limit.

3. The method according to claim 2, further comprising the step of calculating an object's age by timing a current period starting when the respective object was instantiated.

4. The method according to claim 2, further comprising the step of calculating an object's instance count growth as the magnitude of growth of an object instance count over a given time period.

5. The method according to claim 2, wherein the step of monitoring comprises monitoring objects within a class designated for monitoring.

6. The method according to claim 2, further comprising the step of performing the current stack walkbacks for the identified objects.

7. The method according to claim 6, further comprising the step of generating a statistics report including current stack walkbacks for the identified objects.

8. The method according to claim 7, further comprising the step of generating a web interface for user viewing of the statistics report at a computer display.

9. The method according to claim 1, wherein the software objects are Java objects.

10. A computer-implemented system for assisting the detection of critical memory leaks in a software program comprising:

means for monitoring an amount of available memory for the software program during execution of the software program;

means for determining if the amount of available memory for the software program is less than a predetermined amount; and

means for, in response to determination, storing a current stack walkback of each object currently referenced by the software program prior to the available memory dropping below an amount necessary to store the current stack walkback, wherein the current stack walkback assists in the defection of critical memory leak during execution of the software program.

11. The system according to claim 10, further comprising:

means for monitoring a specified one or more analysis properties of the objects referenced by the software program, wherein the one or more specified analysis properties consists of at least one of an object's age and an object's instance count;

means for determining if any analysis property of the objects being referenced following a garbage collection process exceeds a predetermined limit for such analysis property, wherein the predetermined limit for an object's age is an object age limit and the predetermined limit for

an object's instance count is an object instance count growth value; and

means for identifying software objects determined to have one or more analysis properties that exceeds that property's predetermined limit.

12. The system according to claim 11, farther comprising means for calculating an object's age by timing a current period starting when the respective object was instantiated.

13. The system according to claim 11, further comprising means for calculating object instance count growth as the magnitude of growth of an object's instance count over a given time period.

14. The system according to claim 11, wherein means for monitoring comprises means for monitoring objects within a class designated for monitoring.

15. The system according to claim 11, further comprising means for performing the current stack walkbacks for the identified objects.

16. The system according to claim 15, further comprising means for generating a statistics report including current stack walkbacks for the identified objects.

17. The system according to claim 16, further comprising means for generating a web interface for user viewing of the statistics report at a computer display.

18. The system according to claim 1, wherein the software objects are Java objects.

19. An article of manufacture comprising machine-readable medium including program logic embedded therein for assisting the detection of critical memory leaks in a software program that causes control circuitry in a data processing system to perform the steps of:

monitoring an amount of available memory for the software program during execution of the software program;

determining if the amount of available memory for the software program is less than a predetermined amount; and

in response to such determination, storing a current stack walkback of each object currently referenced by the software program prior to the available memory dropping below an amount necessary to store the current stack walkback, wherein the current stack walkback assists in the detection of a critical memory leak during execution of the software program.

20. The article of manufacture of Claim 19, further comprising the steps of:

monitoring. a specified one or more analysis properties of the objects referenced by the software program, wherein the one or more specified analysis properties consists of at least one of an object's age and an object's instance count;

determining if any analysis if any analysis property of the objects being referenced following a garbage collection process exceeds a predetermined limit for such analysis property, wherein the predetermined limit for an object's age is an object age limit and the predetermined limit for an object's instance count is an object instance count growth value; and

identifying any objects determined to have one or more analysis properties that exceeds that property's predetermined limit.

21. The article of manufacture of Claim 20, further comprising the step of calculating an object's age by timing a current period starting when the respective object was instantiated.

22. The article of manufacture of Claim 20, further comprising the step of calculating object instance count growth as the magnitude of growth of an object's instance count over a given time period.

23. The article of manufacture of Claim 20, wherein the step of monitoring comprises monitoring objects within a class designated for monitoring.

24. The article of manufacture of Claim 20, further comprising the step of performing the current stack walkbacks for the identified objects.

25. The article of manufacture of Claim 24, further comprising the step of generating a statistics report including current stack walkbacks for the identified objects.

26. The article of manufacture of Claim 25, further comprises the step of generating a web interface for user viewing of the statistics report at a computer display.

27. The article of manufacture of Claim 19, wherein the software objects are Java objects.

EVIDENCE APPENDIX

Other than the Office Action(s) and reply(ies) already of record, no additional evidence has been entered by Appellants or the Examiner in the above-identified application which is relevant to this appeal.

RELATED PROCEEDINGS APPENDIX

There are no related proceedings as described by 37 C.F.R. §41.37(c)(1)(x) known to Appellant, Appellant's legal representative, or assignee.